PRAKTIKUM 12
ALGORITMA PENGURUTAN (SHELL SORT)

A. TUJUAN PEMBELAJARAN

1. Memahami step by step algoritma pengurutan *shell sort.*
2. Mampu mengimplementasikan algoritma pengurutan *shell sort* dengan berbagai macam parameter berupa tipe data primitif atau tipe Generic.
3. Mampu mengimplementasikan algoritma pengurutan *shell sort* secara ascending dan descending.

B. DASAR TEORI

Shell Sort

Metode ini disebut juga dengan metode pertambahan menurun (diminishing increment). Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959, sehingga sering disebut dengan Metode Shell Sort. Metode ini mengurutkan data dengan cara membandingkan suatu data dengan data lain yang memiliki jarak tertentu, kemudian dilakukan penukaran bila diperlukan

Proses pengurutan dengan metode Shell dapat dijelaskan sebagai berikut :
Pertama-tama adalah menentukan jarak mula-mula dari data yang akan dibandingkan, yaitu N / 2. Data pertama dibandingkan dengan data dengan jarak N / 2. Apabila data pertama lebih besar dari data ke N / 2 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama yaitu N / 2. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-j selalu lebih kecil daripada data ke-(j + N / 2).

Pada proses berikutnya, digunakan jarak (N / 2) / 2 atau N / 4. Data pertama dibandingkan dengan data dengan jarak N / 4. Apabila data pertama lebih besar dari data ke N / 4 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan

dengan jarak yang sama yaitu N / 4. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-j lebih kecil daripada data ke-(j + N / 4).

Pada proses berikutnya, digunakan jarak (N / 4) / 2 atau N / 8. Demikian seterusnya sampai jarak yang digunakan adalah 1.

Algoritma metode Shell dapat dituliskan sebagai berikut :

```
1     Jarak ← N
2     Selama (Jarak > 1) kerjakan baris 3 sampai dengan 9
3     Jarak ← Jarak / 2.  Sudah ← false
4     Kerjakan baris 4 sampai dengan 8 selama Sudah = false
5     Sudah ← true
6     j ← 0
7     Selama (j < N – Jarak) kerjakan baris 8 dan 9
8     Jika (Data[j] > Data[j + Jarak] maka tukar Data[j], Data[j
      + Jarak].  Sudah ← true
9     j ← j + 1
```

Untuk lebih memperjelas langkah-langkah algoritma penyisipan langsung dapat dilihat pada tabel 1. Proses pengurutan pada tabel 1 dapat dijelaskan sebagai berikut:

- Pada saat Jarak = 5, j diulang dari 0 sampai dengan 4. Pada pengulangan pertama, Data[0] dibandingkan dengan Data[5]. Karena 12<17, maka tidak terjadi penukaran. Kemudian Data[1] dibandingkan dengan Data[6]. Karena 35>23 maka Data[1] ditukar dengan Data[6]. Demikian seterusnya sampai j=4.
- Pada saat Jarak = 5/2 = 2, j diulang dari 0 sampai dengan 7. Pada pengulangan pertama, Data[0] dibandingkan dengan Data[2]. Karena 12>9 maka Data[0] ditukar dengan Data[2]. Kemudian Data[1] dibandingkan dengan Data[3] juga terjadi penukaran karena 23>11. Demikian seterusnya sampai j=7. Perhatikan untuk Jarak = 2 proses pengulangan harus dilakukan lagi karena ternyata Data[0] > Data[2]. Proses pengulangan ini berhenti bila Sudah=true.
- Demikian seterusnya sampai Jarak=1.

Tabel 1. Proses Pengurutan dengan Metode Shell

Iterasi	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]	Data[9]
Awal	12	35	9	11	3	17	23	15	31	20
Jarak=5	12	35	9	11	3	17	23	15	31	20
i=6	12	**35**	9	11	3	17	**23**	15	31	20
Jarak=2	12	23	9	11	3	17	35	15	31	20
i=2	**12**	23	**9**	11	3	17	35	15	31	20
i=3	9	**23**	12	**11**	3	17	35	15	31	20
i=4	9	11	**12**	23	**3**	17	35	15	31	20
i=5	9	11	3	**23**	12	**17**	35	15	31	20
i=7	9	11	3	17	12	**23**	35	**15**	31	20
i=8	9	11	3	17	12	15	**35**	23	**31**	20
i=9	9	11	3	17	12	15	31	**23**	35	**20**
i=2	**9**	11	**3**	17	12	15	31	20	35	23
i=3	3	**11**	9	**17**	12	15	31	20	35	23
Jarak=1	3	11	9	15	12	17	31	20	35	23
i=2	3	**11**	**9**	15	12	17	31	20	35	23
i=4	3	9	11	**15**	**12**	17	31	20	35	23
i=7	3	9	11	12	15	17	**31**	**20**	35	23
i=9	3	9	11	12	15	17	20	31	**35**	**23**
i=1	**3**	**9**	11	12	15	17	20	31	23	35
i=8	3	9	11	12	15	17	20	**31**	**23**	35
i=9	3	9	11	12	15	17	20	23	**31**	**35**
Akhir	3	9	11	12	15	17	20	23	31	35

Di bawah ini merupakan prosedur yang menggunakan metode Shell.

```
void ShellSort(int N)
{
   int Jarak, i, j;
   bool Sudah;
   Jarak = N;
   while(Lompat > 1){
      Jarak = Jarak / 2;
      Sudah = false;
      while(!Sudah){
         Sudah = true;
         for(j=0; j<N-Jarak; j++){
            i = j + Jarak;
```

```
            if(Data[j] > Data[i]){
               Tukar(&Data[j], &Data[i]);
               Sudah = false;
            }
         }
      }
   }
}
```

Program 1. Prosedur Pengurutan dengan Metode Shell

C. TUGAS PENDAHULUAN

Jelaskan algoritma pengurutan *shell sort* secara *ascending* dengan data 5 6 3 1 2

D. PERCOBAAN

Percobaan 1 : *Shell sort* secara *ascending* dengan data int.

```
public class ShellDemo {

    public static void shellSort(int[] arr) {
        int n = arr.length;
        int C,M ;

        int jarak, i, j, kondisi;
        boolean Sudah = true;
        int temp ;

        C = 0;
        M = 0;
        jarak = n;

        while (jarak >= 1) {
            jarak = jarak / 2;
            Sudah = true;
            while (Sudah) {
                Sudah = false;
                for (j = 0; j < n - jarak; j++) {
                    i = j + jarak;
                    C++;
                    if (arr[j]> arr[i]) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                        Sudah = true;
                    }
                }
```

```
            }
        }

    }
     public static void tampil(int data[]){
         for(int i=0;i<data.length;i++)
             System.out.print(data[i]+" ");
         System.out.println();
    }
}

public class MainShell {
    public static void main(String[] args) {
        int A[] = {10,6,8,3,1};
        ShellDemo.tampil(A);
        ShellDemo.shellSort(A);
        ShellDemo.tampil(A);
    }
}
```

Percobaan 2 : *Shell sort* secara *descending* dengan data int.

```
public class ShellDemo {

    public static void shellSort(int[] arr) {
        int n = arr.length;
        int C,M ;

        int jarak, i, j, kondisi;
        boolean Sudah = true;
        int temp ;

        C = 0;
        M = 0;
        jarak = n;

        while (jarak >= 1) {
            jarak = jarak / 2;
            Sudah = true;
            while (Sudah) {
                Sudah = false;
                for (j = 0; j < n - jarak; j++) {
                    i = j + jarak;
                    C++;
                    if (arr[j]> arr[i]) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                        Sudah = true;
                    }
                }
            }
```

```
        }
    }
     public static void tampil(int data[]){
         for(int i=0;i<data.length;i++)
             System.out.print(data[i]+" ");
         System.out.println();
    }
}

public class MainShell {
    public static void main(String[] args) {
        int A[] = {10,6,8,3,1};
        ShellDemo.tampil(A);
        ShellDemo.shellSort(A);
        ShellDemo.tampil(A);
    }
}
```

Percobaan 3 : *Shell sort* secara *ascending* dengan data double.

```
public class ShellDemo {
    public static void shellSort(double[] arr) {
        int n = arr.length;
        int C,M ;

        int jarak, i, j, kondisi;
        boolean Sudah = true;
        double temp ;

        C = 0;
        M = 0;
        jarak = n;

        while (jarak >= 1) {
            jarak = jarak / 2;
            Sudah = true;
            while (Sudah) {
                Sudah = false;
                for (j = 0; j < n - jarak; j++) {
                    i = j + jarak;
                    C++;
                    if (arr[j]> arr[i]) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                        Sudah = true;
                    }
                }
            }
        }

    }
     public static void tampil(double data[]){
```

```
         for(int i=0;i<data.length;i++)
             System.out.print(data[i]+" ");
         System.out.println();
    }
}

public class MainShell2 {
    public static void main(String[] args) {
        double A[] = {10.3,6.2,8.4,3.6,1.1};
        ShellDemo.tampil(A);
        ShellDemo.shellSort(A);
        ShellDemo.tampil(A);
    }
}
```

E. LATIHAN

1. Buatlah program sorting Shell dengan parameter array Integer (class Wrapper) !

```
public static void shellSort(Integer[] A){…}
```

2. Buatlah program sorting Shell dengan parameter array Double (class Wrapper) !

```
public static void shellSort(Double[] A){…}
```

3. Buatlah fungsi tampil() untuk menampilkan data.

```
public static<T> void tampil(T data[]){ }
```

4. Lakukan pengujian fungsi shellSort(), dengan membuat fungsi main() sebagai berikut :

```
public class Demo1 {
    public static void main(String[] args) {
        //Data Integer
        Integer arr3[] = {1,5,6,2,8,9};
        ShellDemo.shellSort(arr3);
        ShellDemo.tampil(arr3);

        //data Double
        Double arr4[] = {1.3,5.2,6.6,2.7,8.8,9.1};
        ShellDemo.shellSort(arr4);
        ShellDemo.tampil(arr4);
    }
}
```

5. Buatlah program sorting Shell dengan parameter array Number !

```
public static<T extends Number> void shellSort(T[] A){…}
```

6. Lakukan pengujian fungsi shellSort(), dengan membuat fungsi main() sebagai berikut :

```
public class Demo2 {
    public static void main(String[] args) {
        Float arr5[] = {1.3f,5.2f,6.6f,2.7f,8.8f,9.1f};
        ShellDemo.shellSort(arr5);
        ShellDemo Demo.tampil(arr5);

        Byte arr6[] = {6,7,11,1,3,2};
        ShellDemo.shellSort(arr6);
        ShellDemo.tampil(arr6);
    }
}
```

7. Buatlah program sorting Shell dengan parameter array yang memenuhi T extends Comparable !

```
public static <T extends Comparable> void shellSort2(T[] arr) {
}
```

8. Lakukan pengujian fungsi shellSort(), dengan membuat fungsi main() sebagai berikut :

```
public class Demo3 {
    public static void main(String[] args) {
        //data String
        String arr7[]=
{"jeruk","anggur","belimbing","jambu","kelengkeng"};
        ShellDemo.shellSort2(arr7);
        ShellDemo.tampil(arr7);
    }
}
```

9. Buatlah class Mahasiswa dengan variable nrp dan nama yang memiliki tipe String ! Class Mahasiswa mengimplementasikan interface Comparable, selanjutnya implementasikan fungsi abstract compareTo(), untuk membandingkan dua objek mahasiswa berdasarkan nrp.

```
public class Mahasiswa implements Comparable <Mahasiswa> {
    private String nrp ;
    private String nama ;
    @Override
    public int compareTo(Mahasiswa o) {…}

    @Override
    public String toString() {…}
}
```

10. Lakukan pengujian fungsi shellSort() lagi, sebelumnya tambahkan pada fungsi main() seperti di bawah ini !

```
public class Demo4 {
    public static void main(String[] args) {
        Mahasiswa arr8[] = {new Mahasiswa("02", "Budi"), new
Mahasiswa("01", "Andi"), new Mahasiswa("04", "Udin"), new
Mahasiswa("03", "Candra")};
        ShellDemo.shellSort 2(arr8);
        ShellDemo.tampil(arr8);
    }
}
```

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.